Hadoop components, YARN and Apache Tez

In this video, we'll learn about a Hadoop resource manager named YARN as well as Tez, a framework based off of it that is running high performance batch and interactive data processing applications in Apache Hadoop.
YARN means Yet Another Resource Negotiator that provides resource management, meaning, CPU, memory, and disks.
And a central platform to deliver consistent operations, security, and data governance tools across Hadoop clusters.
In 2012, YARN became a subproject of the larger Apache Hadoop project, sometimes called MapReduce 2.0.
YARN takes care of common issues, such as running apps on shared resources including scheduling, contention, starvation, pre-emption, etc.
It is a general purpose resource manager that supports a variety of application types such as Distributed Shell, Impala, Apache Giraph, Spark, and MapReduce.
There are some limitations of pre-YARN architecture.
YARN was created specifically to address these limitations.
With the pre-Yarn architecture, you have limits to scalability, JobTracker runs on a single machine during several task like resource management, job and task scheduling and monitoring.
Although there are so many machines or data nodes available, they are not getting used, which limits scalability.
You also have a problem with resource utilization.
In Hadoop 1.0, there is a concept of predefined number of map slots and reduce slots for task trackers.
Resource utilization issues occur because map slots might be full while reduce slots are empty and vice-versa.
Here are the compute resources, data nodes could sit idle which are reserved for reduce slots even when there's immediate need for those resources to be used as mapper slots.
There's also a limitation in running non-MapReduce applications.
In Hadoop 1.0 JobTracker was tightly integrated with MapReduce and only supporting applications that obey MapReduce programming framework that run can on Hadoop.
Finally, you will an unavailability issue.
In Hadoop 1.0, JobTracker is a single point of availability, this means if JobTracker fails, all jobs must restart.
YARN addresses these limitations.
Here are some advantages of YARN.
YARN has an efficient utilization of the resource.
There are no more fixed MapReduce slots.
YARN provides essential resource manager.

And with YARN, you can now run multiple applications on Hadoop, all sharing a common resource.
YARN can even run applications that do not follow the MapReduce model.
YARN decouples MapReduce's resource management and scheduling capabilities from the data processing components, enabling Hadoop to support more varied processing approaches in a broader array of applications.
For example, Hadoop clusters can now run interactive querying and streaming data applications simultaneously with MapReduce batch jobs.
This also streamlines MapReduce to do what it does best, process data.
Here are the roles and responsibilities of each and every YARN component.
At the root of a YARN hierarchy, is a resource manager.
This entity governs an entire cluster, and manages the assignment of applications to underlying compute resources.
The resource manager then orchestrates the division of resources, the compute, memory, bandwidth, etc., to underlying node managers, YARN's per node agent.
The node manager manages each node within a YARN cluster.
The node manager provides per node services within the cluster from overseeing the management of a container over its life cycle to monitoring resources and tracking node health.
Containers considered to be a small unit of resources like CPU and memory and disks, belong to the detail node.
Schedule or processes running along with the resource manager daemon allocates resources as a container.
At the beginning of a job execution with YARN, container allows the application master process to make use of some resources on any detail node within the cluster.
Then the application manager manages the application execution across other containers on detail nodes of a YARN.
Its per application specific library, works with node manager to execute the task.
The instance of the daemon is per application, which means that in the case of multiple jobs submitted on the cluster, it may have more than one instance of application manager on detail nodes.
In the flow depicted on the slide, you can see how components are arranged in a typical YARN cluster.
Use of a YARN cluster begins with a request from a client consisting of an application.
The resource manager then negotiates the necessary

resources for a container and launches an application manager
to represent the submitted application.
Using a resource request protocol, the application
manager negotiates resource containers for
the applications at each node.
Upon execution of the application,
the application manager monitors the container until completion.
When the application is complete,
the application manager unregisters its container with
the resource manager and the cycle is complete.
The job or application which can be MapReduce, Java,
Scala application, DAG jobs like Apache Spark, etc.,
is submitted by the YARN client application to the resource
manager daemon along with the command that start
the application master on any container at node manager.
The application manager process on master node
validates the job submission request and
hands it over to the scheduler process for resource allocation.
Scheduler process then assigns a container for
the application manager on one detail node.
The node manager daemon starts the application master
service within one of its containers
using the command mentioned in step 1.
Hence, application manager is considered to be
the first container of any application.
The application manager negotiates the other containers
from resource manager by providing the details
like the location of the data on the detail node, required CPU,
memory, cores, etc.
Resource manager allocates the best suitable resources
on detail node and responds to the application master
with node details and other details.
Then, the application manager sends request to the node
managers on suggested detail nodes to start the containers.
The application master then manages the resources
of the requested containers during the job execution and
notifies the resource manager when the execution is completed.
Node managers periodically notify the resource manager
with the current status of available resources on the node.
That information can be used by scheduler to schedule new
applications on the clusters.
In case of any failure of the detail node,
resource manager will try to allocate new containers
on other best suitable nodes so that the application manager
can complete the process using a new container.
Apache Tez provides a developer API and

framework to write native YARN applications
that bridge the spectrum of interactive and batch workloads.
It allows those data access applications to work with
petabytes of data, over thousands of notes and
expresses computation as a dataflow graph or DAG.
The application Tez component library,
allows developers to create Hadoop applications that
integrate natively with the Apache Hadoop YARN and
perform well within mixed workload clusters.
Some of the benefits are, since Tez is extendable and
embeddable, it provides the fit to purpose freedom
to express highly optimized data processing applications,
giving them an advantage over end user facing engines,
such as MapReduce and Apache Spark.
It also supports a wider range of applications than MapReduce.
Tez offers a customizable execution architecture that
allows users to express complex computations as dataflow graphs,
permitting dynamic performance optimizations
based on real information about the data and
resources required to process it.
Tez provides an expressive data flow definition API, so
that you can describe the directed acyclic graph,
DAG, of computation that you want to run.
For this, Tez has a structural kind of API
in which you can add all processors and edges and
visualize what you are actually constructing.
Tez models data processing has a dataflow graph, with the graph
vertices representing application logic and
it's edges representing movement of data.
A rich data flow definition API
allows users to intuitively express complex queries logic.
The API fits well with query plans produced by higher level
declarative applications like Apache Hive and Apache Pig.
Tez defines a simple JAVA API
to express a DAG of data processing.
The API has three components, DAG,
which defines the overall job.
The user creates a DAG object for each data processing job.
Then you have vertex, which defines the user logic and
the resources and
environment needed to execute the user logic.
The user creates a vertex object for each step in the job and
adds it to the DAG.
Then there's the edge, which defines the connection
between the producer and consumer vertices.
The user creates an edge object and connects the producer and

consumer vertices using it.
The data processing in Tez is expressed in the form of
a directed acyclic graph, or DAG.
The processing starts at the root vertices of the DAG and
continues down the directed edges
until it reaches the leaf vertices.
When all of the vertices in the DAG have been completed,
then the data processing job is done.
The diagram shows a dataflow graph and
its definition using the DAG API, which is simplified.
The job consists of two vertices performing a map operation
on two datasets.
Their output is consumed by two vertices
that do a reduce operation.
Their output is brought together in the last vertex that does
a join operation.
Tez handles expanding this logical graph at run-time
to perform the operations in parallel using multiple tasks.
The diagram shows a run-time expansion
in which the first MR pair has a parallelism
of 2 while the second has a parallelism of 3.
Both branches of computation merge in the join operation
that has a parallelism of 2.
Edge properties are at the very heart of this run-time activity.
Tez provides a single underlined framework to support
both latency and through put sensitive applications,
thereby obviating the need for multiple frameworks and
systems to be installed, maintained and supported.
A key advantage to enterprises looking to
rationalize their data architectures.
This helps in the following ways.
It eliminates the replicated right barrier between successive
computations, and
it eliminates the job launch overhead of workflow jobs.
It also eliminates an extra stage of map reads in
every workflow job.
It eliminated queue and resource contentions suffered by workflow
jobs that are started after a predecessor job completes.
It also achieves better locality because the engine has
the global picture.
In a task, after the test framework has initialized
all of the necessary inputs, outputs, and the processor,
the Tez framework invokes the processor's run function and
passes the appropriate handles to all the inputs and
outputs for that particular task.
Thus, the application can modify the DAG at run-time

to optimize execution.
Applications can also change the task concurrency,
change the task scheduling, and change DAG edges.